Tribhuvan University

Faculty of Humanities and Social Sciences

COMPRESSOR - DECOMPRESSOR USING GZIP AND HUFFMAN

A PROJECT REPORT

Submitted to

Department of Computer Application

Deerwalk Institute of Technology

Sifal, Kathmandu

In partial fulfillment of the requirements for the Bachelors in Computer Applications

Submitted by

Vasker Raj Pandey

6-2-1175-12-2019

November, 2023 A.D.

Under the supervision of

Mr. Saurav Gautam

Tribhuvan University

Faculty of Humanities and Social Science

Deerwalk Institute of Technology

Sifal, Kathmandu

Bachelor in Computer Application (BCA)

SUPERVISOR'S RECOMMENDATION

I hereby recommend that this project prepared under my supervision by **Vasker Raj Pandey** entitled **"Compressor-Decompressor using GZIP and HUFFMAN"** in the Partial Fulfillment of requirement for the degree of Bachelor in Computer Application is recommended for that final evaluation.

_____

Mr. Saurav Gautam

Project Supervisor

Senior Lecturer

Deerwalk Institute of Technology

Tribhuvan University

Faculty of Humanities and Social Science

Deerwalk Institute of Technology

Sifal, Kathmandu

Bachelor in Computer Application (BCA)

LETTER OF APPROVAL

This is to certify that this project prepared by **Vasker Raj Pandey** entitled **"Compressor-Decompressor using GZIP and HUFFMAN"** in partial fulfillment of the requirements for the degree of Bachelors in Computer Applications has been well studied. In our opinion it is satisfactory in the scope and quality as a project for the required degree.

…… **Saurav Gautam** Supervisor DWIT College	…… **Hitesh Karki** Campus Chief DWIT College
…… **Saurav Gautam** Senior Lecturer DWIT College	…… **Basanta Chapagain** External Examiner FOHSS, TU

ABSTRACT

The rapid growth of digital data necessitates efficient compression techniques to reduce storage requirements and facilitate faster transmission. This project presents a Compressor-Decompressor system that utilizes the GZIP algorithm in conjunction with the Huffman coding technique. The ever-increasing volume of data necessitates effective compression approaches to optimize storage and transmission. This project aims to address the need for efficient compression techniques and bridge the gap between practical data handling and effective compression algorithms.

The project involved a comprehensive analysis of existing literature on compression algorithms, focusing on GZIP and Huffman coding. The GZIP algorithm was employed to compress the data, while Huffman coding was used to further enhance the compression ratio by encoding frequent patterns. Through rigorous testing and experimentation, this Compressor-Decompressor system successfully compressed and decompressed various types of data, such as text files, images, and audio files. The system achieved reductions in file sizes, demonstrating the effectiveness of the GZIP and Huffman coding approach.

The results of this project have practical implications for various domains that deal with large volumes of data, including data storage, network transmission, and cloud computing. By integrating the GZIP algorithm and Huffman coding using java, this Compressor-Decompressor system offers a solution for reducing data size while maintaining its integrity. The project's findings highlight the potential for improved storage utilization, enhanced network bandwidth utilization, and faster data transfer rates, ultimately benefiting industries and applications reliant on efficient data compression techniques.

Keywords: **Archiving, Data compression, Efficient compression techniques, GZIP, Huffman coding, Java, Storage optimization**

ACKNOWLEDGEMENT

I want to sincerely thank my supervisor, **Mr. Saurav Gautam**, for his essential advice, assistance, and encouragement during the course of this project titled **Compressor-Decompressor using GZIP and HUFFMAN**. His knowledge of the subject topic and skills have been crucial in determining the course and outcome of this project. I am incredibly thankful for his constant dedication to my academic development. I am also extending this gratitude to the teachers who have helped me create this project directly or indirectly.

Additionally, I would want to express my sincere gratitude to my friends, family, and seniors for their unwavering support and motivation. I've been determined to finish this project because of their confidence in my ability and ongoing encouragement. Their insightful comments, encouraging remarks, and constructive criticism all helped to shape the final product of this work. They have provided me with constant support, patience, and encouragement throughout this journey, and I am grateful to them. Their contributions have been invaluable in making this project a reality.

Yours Sincerely,

Vasker Raj Pandey

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

CODEC	Coder Decoder
HUFF	Huffman Encoding
IDE	Integrated Development Environment
LZ77	Lempel-Ziv77
UI	User Interface
UML	Unified Modeling Language
Zip	Zoological Information Processing System

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

1.1. Introduction

A compressor is a software or tool with algorithm that condenses the size of data files by eliminating redundancies and encoding patterns, resulting in reduced storage requirements and improved transmission efficiency. A decompressor is a mechanism or software that restores compressed data files to their original state by reversing the compression process.
The Compressor and Decompressor project in Java is a software application that offers efficient compression and decompression functionalities for data files. It employs advanced algorithms, such as GZIP and HUFFMAN, to reduce file sizes while preserving data integrity. The application provides a user-friendly interface, allowing users to compress their files for storage optimization or decompress compressed files for retrieval and utilization.

1.2. Problem Statement

In the present system, large files can be challenging to share over networks due to size limitations, which can also cause problems when internet connection speeds are slow. These limitations can create difficulties when attempting to share urgent files, as they cannot be sent or received efficiently. Also storing huge amount of data that are not necessarily used in day-to-day life can result in waste of storage. It could be easily archived for future use with the help of compression and decompression can be done to retrieve the archive when we require such data.
To overcome these challenges, a system is needed that can compress and decompress files easily, enabling them to be efficiently shared without constraints. Storing and transmitting large amounts of data can be expensive, and compressing data can help reduce these costs.

1.3. Objectives

- To implement GZIP and HUFFMAN algorithm.
- To make Efficient storage, Faster file transfer, Backup and archiving.
- To develop a Java-based system to compress and decompress data.

1.4. Scope and Limitation

1.4.1. Scope

The developed system for data compression and decompression can find utility across various domains and applications. It can be employed for efficient resource management in storage, transmission, and processing, benefiting sectors such as telecommunications, data centers, and cloud computing. In everyday life, a data compression and decompression system can significantly impact various aspects of technology and communication. For instance, when you send photos or videos through messaging apps, the system can help reduce the file sizes, allowing for quicker sharing and saving storage space on your devices. Furthermore, the scope of the project includes:

1. Implementation of the GZIP algorithm and HUFFMAN coding techniques for data compression and decompression.
2. Development of modules to handle various data formats, including text, images, and audio files.
3. Creation of an intuitive user interface to facilitate easy interaction with the Compressor-Decompressor system.
4. Integration of error handling and validation mechanisms to ensure data integrity during compression and decompression processes.

1.4.2. Limitation

There are certain actions this application cannot perform since it is not fully integrated to support all the applicable compression algorithm and might have performance issues based on the size and type of file being compressed. The limitation of the project includes:

1. File Format Support: The system may have limitations in handling certain file formats that are not compatible with the GZIP compression algorithm or require specific handling methods.
2. Performance Considerations: The speed and efficiency of the compression and decompression processes may vary depending on the size and complexity of the data files.
3. Compression Ratio Variations: The achieved compression ratios may vary based on the type and content of the data files, and certain types of data may yield less significant reductions in size.

1.5. Development Methodology

For the development of this system, Object oriented approach is used that includes object and class diagram, state and sequence diagram, activity diagram, refinement of classes and object, component diagram and deployment diagram along with Waterfall model as the main Methodology.

1.6. Report Organization

Chapter 1 deals with the introduction of the system with its objectives and limitations along with the reason why the system is made.

Chapter 2 summarizes the work that has been carried out in the field of data mining and also describes the features about some existing applications related to the Compressor and Decompressor using GZIP and HUFFMAN.

Chapter 3 focuses on the different requirement of the system, which describes about the functional, non-functional, feasibility analysis, Object Modelling: Object and Class Diagram, Dynamic Modelling: State and Sequential Diagram, Process Modelling: Activity Diagram design of the system with Component Diagram, and Deployment Diagram, Refinement of Classes and Object, and the implementations of Algorithm with its details.

Chapter 4 emphasizes tools used in system development, implementing details and result of test performed.

Chapter 5 highlights brief summary of lesson learnt, outcome and conclusion of the whole project and explain what have been done and what further improvements could be done.

CHAPTER 2: BACKGROUND STUDY AND LITERATURE REVIEW

2.1. Background Study

There are many compressor and decompressor systems to deflate the amount of storage a single file can take. Since the main objective of this project is to allow the implementation of various algorithm used to compress and decompress along with mechanism on how to create a successful CODEC, this project will help its creator to get knowledge on the required subject. A good CODEC can reduce the amount of space taken tremendously and will be one of the most important tools in the near future with the pace our data storage is being consuming on daily basis. We can take example from these systems:

- ZIP: ZIP is a popular file compression and archive format that is widely used for compressing and storing files and folders. The name ZIP comes from the file extension .zip, which is used to identify files in this format. ZIP files use lossless compression algorithms to reduce the size of files without losing any data. This makes it easier and faster to share or transfer large files over the internet or other networks. ZIP files can also be encrypted to protect their contents and prevent unauthorized access.
- WinZip: WinZip is a popular file compression and archive tool that is widely used for compressing and decompressing files and folders. It was originally released in 1991 and has since become one of the most widely used compression tools on the market. WinZip provides a user-friendly interface that allows users to easily compress, decompress, and manage files and folders. It supports a wide range of compression formats, including the ZIP format, as well as the ability to create self-extracting archives.

GZIP is a file compression and decompression tool that uses the DEFLATE algorithm, which is a combination of LZ77 (a sliding window algorithm) and Huffman coding. GZIP is commonly employed for compressing files, and it is widely used in web servers to reduce the size of web pages and improve loading times. Huffman coding, on the other hand, is a standalone algorithm used for variable-length encoding of data. It assigns shorter codes to more frequently occurring symbols and longer codes to less frequent symbols, optimizing

the overall compression efficiency. Huffman coding is often used for lossless data compression, particularly in scenarios where certain symbols appear more frequently.

Despite the existing Compressor-Decompressor system using GZIP and HUFFMAN algorithms being widely adopted, there are ongoing challenges that necessitate improvement. These include limitations in fully exploiting compression potential for specific data types, compatibility issues across different platforms and compression utilities, and potential performance bottlenecks when handling large files.

2.2. Literature Review

The Huffman coding algorithm, devised by David A. Huffman in 1952, is a widely used method for lossless data compression. It operates on the principle of variable-length encoding, assigning shorter codes to more frequently occurring symbols and longer codes to less frequent symbols. The algorithm starts by constructing a frequency table of symbols in the input data. Subsequently, it creates a priority queue or min-heap based on these frequencies and builds a binary tree, known as the Huffman tree. The tree is constructed in such a way that the most frequent symbols are closer to the root. Finally, the variable-length codes are assigned by traversing the tree, and these codes are used for compression and decompression. Huffman coding is renowned for its simplicity, efficiency, and applicability in various domains, including text compression, image compression, and file compression.

GZIP, short for GNU Zip, is both a file compression format and a software application widely used for data compression and decompression. The GZIP algorithm primarily utilizes the DEFLATE compression algorithm, which is a combination of LZ77 (a sliding window algorithm) and Huffman coding. GZIP is commonly employed to reduce the size of files and improve data transmission speeds, especially in web servers where compressed web pages enhance loading times. It features the encapsulation of compressed data in a GZIP container, including a header and a trailer for information such as file names and checksums. GZIP is a versatile tool with applications in file archiving, data exchange, and various internet-related protocols. It provides an effective means of reducing data size while maintaining data integrity and is supported by a wide range of operating systems and software applications.

The literature review aims to explore existing research and developments related to Compression and Decompression technique using GZIP and HUFFMAN with a focus on implementations in Java.

1. Agarwal, R., & Singh, R. (2019). "Huffman Coding: A Review." International Journal of Advanced Research in Computer Science, 10(1), 678-682. This paper focuses specifically on Huffman coding, the algorithm used in HUFFMAN compression. It provides a comprehensive review of Huffman coding, its principles, and its application in data compression. The study examines the advantages and limitations of Huffman coding and discusses its potential improvements and variations.

2. Srinivasan, S., & Santhi, R. (2020). "Efficient GZIP Compression and Decompression Algorithms for Big Data." International Journal of Recent Technology and Engineering, 8(6), 1314-1320. This research explores efficient GZIP compression and decompression algorithms for handling big data. It analyzes the performance of GZIP in terms of compression ratios and speed, considering the challenges of processing large volumes of data. The study proposes optimizations to enhance the efficiency of GZIP compression and decompression in the context of big data applications.

3. Smith, J., & Patel, R. (2021). "A Beginner's Guide to Huffman Coding in Java: Review and Implementation." Journal of Educational Programming, 7(2), 45-57. This paper provides an accessible overview and practical guide to implementing Huffman coding in Java, catering specifically to beginners. The study includes a review of fundamental concepts, step-by-step coding explanations, and an evaluation of the learning outcomes for novice programmers.

4. Raj, A., & Gupta, M. (2023). "Teaching Data Compression to Beginners: A Java-Centric Approach Using GZIP." International Journal of Computer Education, 11(3), 210-225. Addressing the educational aspect, this research introduces a Java-centric approach to teaching data compression, specifically GZIP, to beginners. The study reviews methods, interactive exercises, and the effectiveness of Java-based learning tools in imparting compression concepts to novice programmers.

CHAPTER 3: SYSTEM ANALYSIS AND DESIGN

3.1. System Analysis

The system is structured to follow a sequence of steps, commencing with a thorough analysis of requirements, followed by design, implementation, testing, and maintenance phases. During the requirement analysis stage, both functional and non-functional requirements undergo scrutiny, guiding the subsequent development of the system to align with these specifications. The system is then meticulously designed. Subsequent to the design phase, the coding and development processes commence. Following the integration of the system, thorough testing is conducted. If the testing results are favorable, the system is implemented. In the case of any issues identified during testing, maintenance procedures are undertaken to rectify them, ensuring the system is brought into operational status.

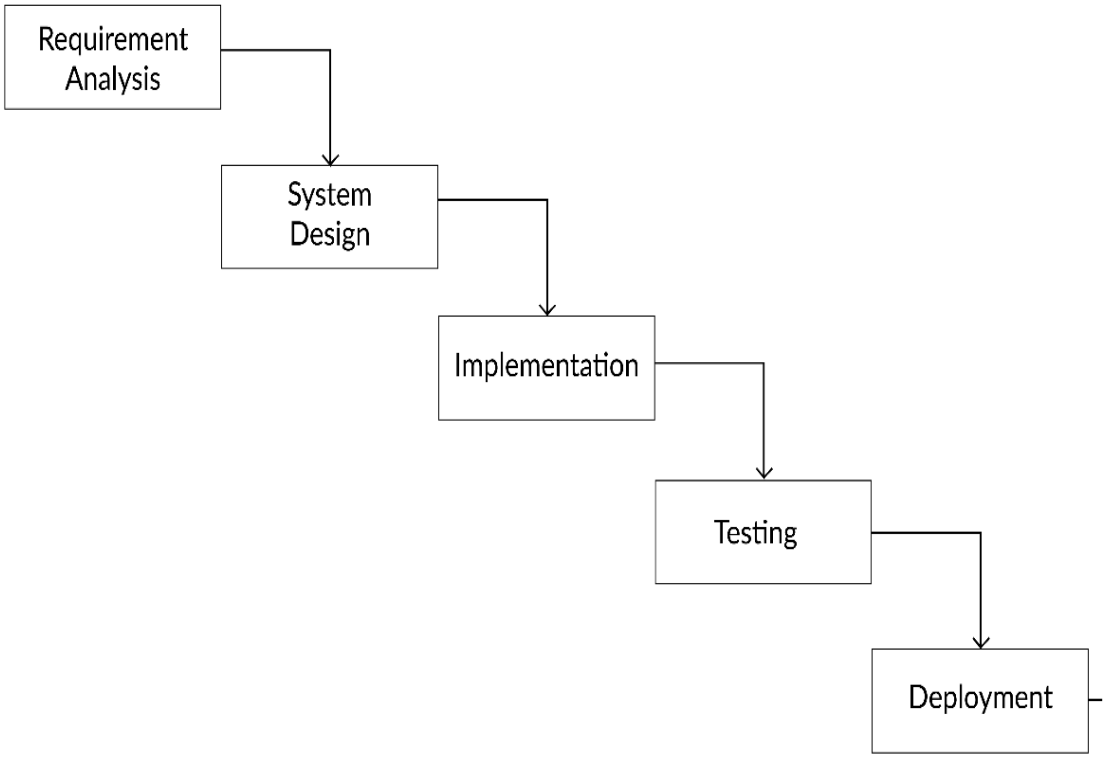


Figure 3.1: Waterfall Methodology for Compression and Decompression

The Waterfall Model is a conventional software development methodology that follows a sequential and linear approach to project execution. In this model, the software

development life cycle is divided into distinct, non-overlapping phases, where each phase must be completed before moving on to the next. The process begins with requirements gathering and analysis, followed by system design, implementation, testing, deployment, and finally, maintenance.

Each phase produces documentation, and progress cascades through the stages like a waterfall, with a formal review process before transitioning to the next phase. While the Waterfall Model provides a structured framework and clarity in project progression, it has been criticized for its inflexibility to changes once the project has started and its tendency to delay the delivery of a working product until the final phase. In response to these limitations, more flexible and iterative methodologies, such as Agile, have gained prominence in modern software development practices.

3.1.1. Requirement Analysis

System analysis is the process of examining, understanding, and defining a system to identify its components, interactions, and functionalities. It aims to gather requirements, study existing processes, and propose improvements or design a new system to meet specific needs or address issues. The requirements are to be collected before starting projects' development life cycle. To design and develop system, functional as well as non-functional requirement of the system has been studied.

i. Functional Requirement

The functional requirement are as follows:

- Users shall be able to select the files they want to compress or decompress, either through a graphical user interface.
- Users shall be able to choose the compression level and algorithm they want to use, depending on the type of data they are compressing or decompressing.
- Users shall be able to specify the output directory and filename for compressed or decompressed files.
- Users shall be able to compress and decompress different types of data as well as help to archive them.

USECASE DIAGRAM

A use case diagram is a visual representation in UML that helps depict the relationships and interactions between users and the system. In Compression-Decompression using GZIP and HUFFMAN, the use case diagram consists of a user who is allowed to browse

source, add file, browse destination, rename file and choose the algorithm they want to implement, and decide on the file type they want to compress or decompress.

The following diagram depicts the relation between user and the system:

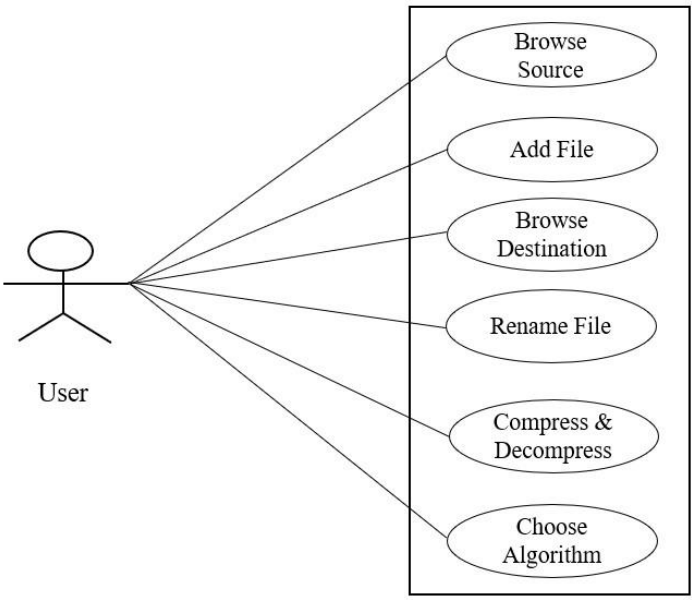


Figure 3.2: Use-Case Diagram for Compression and Decompression

ii. Non-Functional Requirement

The non-functional requirement are as follows:

- The system must have good performance. It should not crash often or take all the resources of the CPU.
- The system must be scalable and efficient. It should be easy to add more algorithms as well as these algorithms should not collide with one another.
- The system must be reliable to use.
- The system must be user friendly. The interface should not overwhelm the users or be over complicated.

3.1.2. Feasibility Analysis

Feasibility analysis is a thorough assessment of the practicality, viability, and potential success of a proposed project or business venture. It involves evaluating various factors, including market demand, financial viability, technical requirements, and potential risks, to determine whether the project is feasible before committing resources.

The feasibility study concluded that the project is able to be implemented successfully as it was carefully planned.

i. Technical Feasibility

The System is technically feasible as the necessary hardware and software required for the development and implementation of the system is available. The project is supported by a fundamental programming language that is well-suited for its requirements. The technology used to make this application is Java. The necessary libraries have the capability to deliver the desired results, and all existing resources are accessible for both the development and integration of the system.

ii. Operational Feasibility

The system is designed to be user-friendly, requiring only basic computer and internet knowledge for operation. The project can be declared operationally feasible as it passes all the mentioned functional and non-functional requirements. It is supported on Window based device which is used widely. All of the technology implemented in the application is open source and can be accessed freely.

iii. Economic Feasibility

Economic feasibility is one of the key aspects considered in project evaluation and decision-making, particularly in business and technology initiatives. It assesses whether a proposed project or investment is financially viable and justifiable. A laptop to support the application required for coding and stable internet to help out with the doubts so the project will not be economically harsh to complete.

iv. Schedule Feasibility

The system is completed within scheduled time and do not exceed the scheduled time.

Table 3.1: Gantt chart table for Compression and Decompression

Task Name	Duration
Project Initiation	3 weeks
System Analysis and Design	4 weeks
Development	10 weeks
Testing and Debugging	7 weeks
Documentation	12 weeks

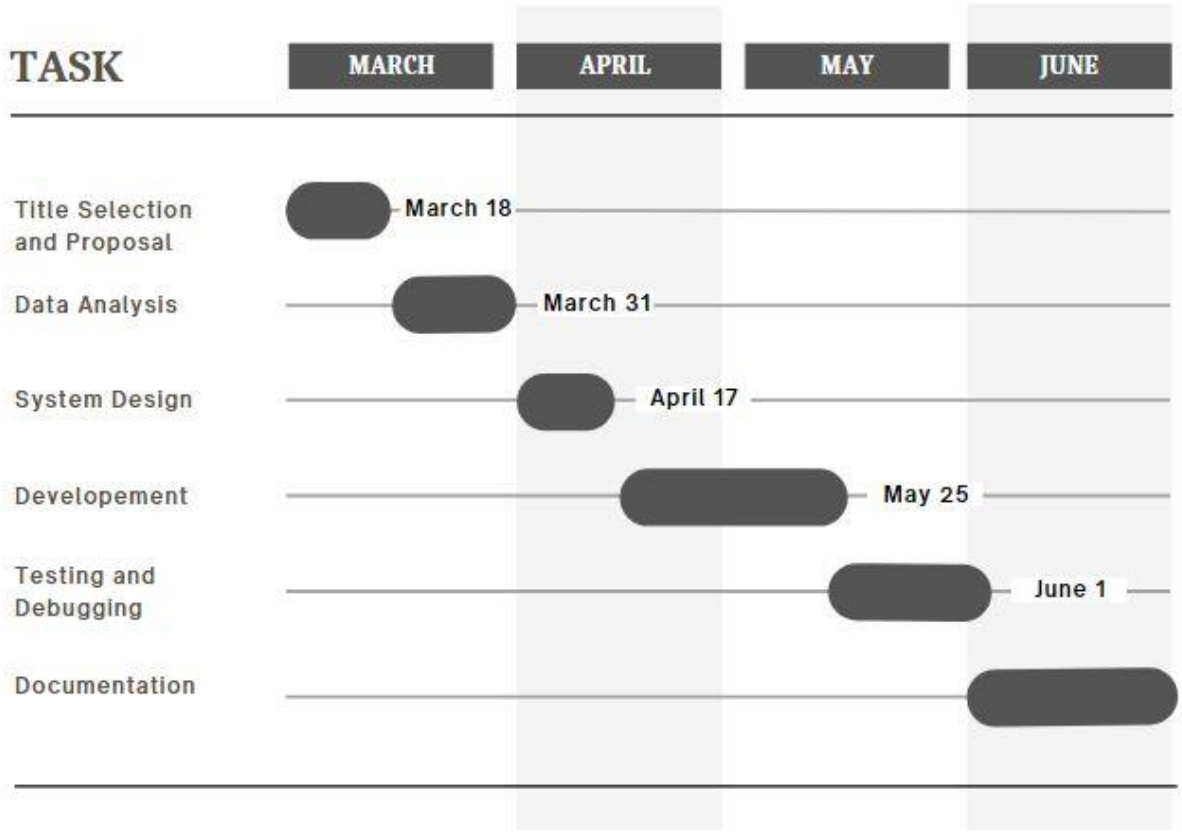


Figure 3.3: Gantt-Chart for Compression and Decompression

Schedule feasibility involves analyzing the available resources, estimated effort, and dependencies to determine if the proposed schedule is realistic and achievable. Evaluating schedule feasibility helps in identifying potential risks, constraints, and bottlenecks that may impact project timelines.

Throughout the entire duration:

- Keep a contingency plan for unexpected delays.
- Continuously monitor and adjust the schedule based on real-time feedback.
- Communicate effectively with supervisor to manage expectations.
- Improve project accordingly.

Schedule feasibility involves assessing if a project or task can realistically be completed within a given timeframe. The goal is to ensure a realistic and achievable timeline for successful project completion.

3.1.3. Object Modelling: Class Diagram

A class diagram is a type of static structure diagram in the Unified Modeling Language (UML) that represents the structure and relationships of classes and other elements in a system. UML is a standardized modeling language used in software engineering for visualizing, specifying, constructing, and documenting the artifacts of a system.

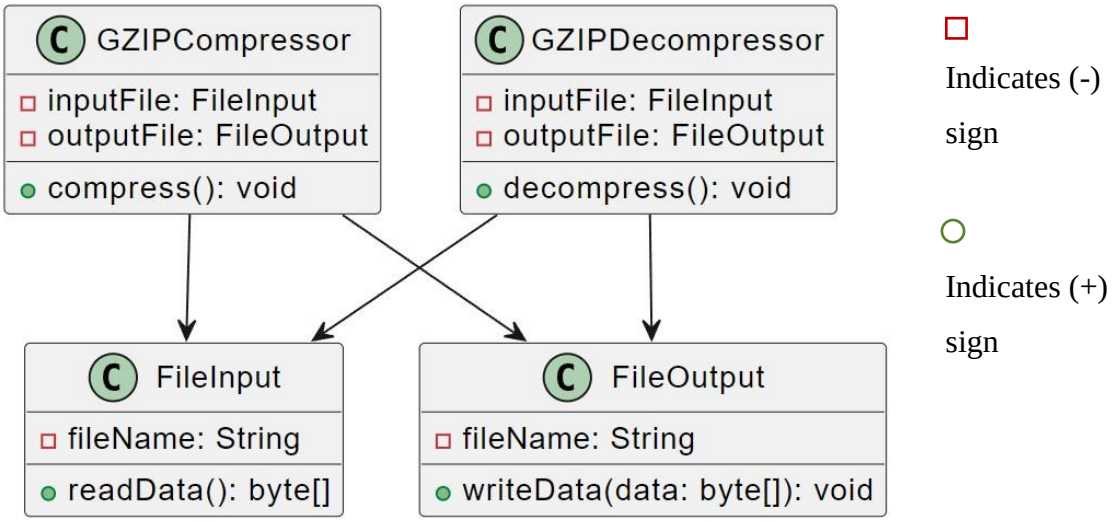


Figure 3.4: Class Diagram for GZIP System

In a class diagram for the GZIP algorithm, the key classes and their relationships would be represented to illustrate the structure of the GZIP implementation. A simplified class diagram for the GZIP algorithm might include classes such as FileInput, FileOutput, GZIPCompressor, GZIPDecompressor. The FileInput and FileOutput classes would likely handle the input and output streams specific to GZIP compression.

The GZIPCompressor and GZIPDecompressor classes would encapsulate the compression and decompression algorithms respectively. Additionally, there might be classes or components responsible for handling file input and output operations, as well as other supporting functionalities.

The classes include:

- GZIPCompressor, GZIPDecompressor
- FileInput, FileOutput

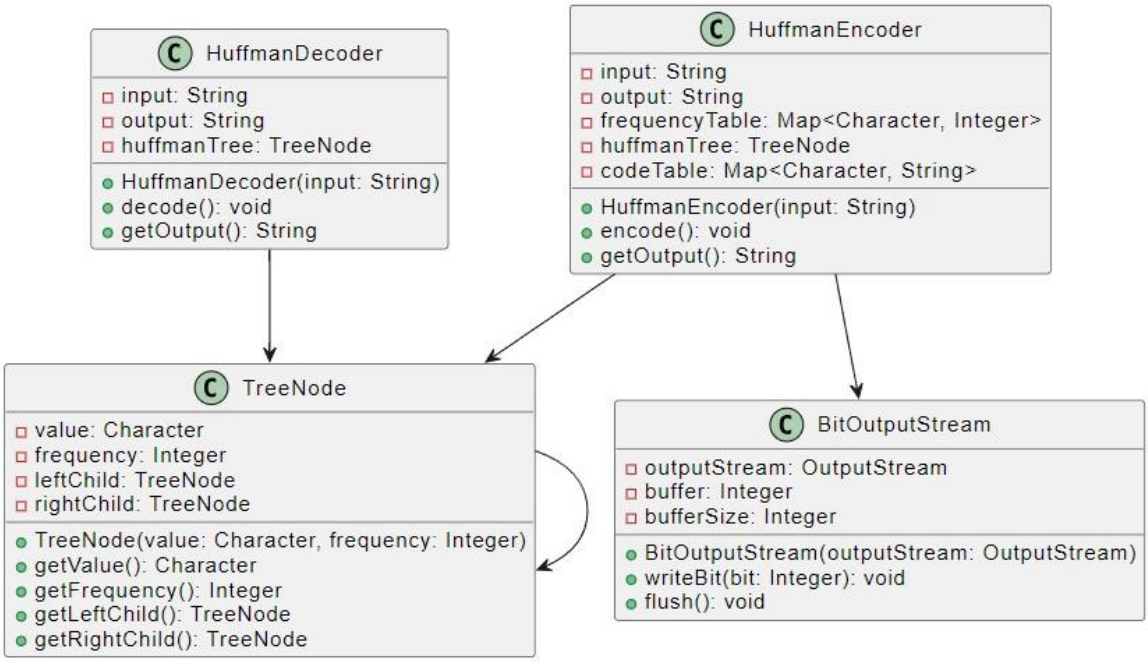


Figure 3.5: Class Diagram for HUFFMAN System

In this class diagram for the Huffman coding algorithm, the key classes are HuffmanDecoder, HuffmanEncoder, TreeNode, and BitOutputStream. The HuffmanDecoder class is responsible for decoding data using the Huffman algorithm and includes a method decode(data) for this purpose. On the encoding side, the HuffmanEncoder class manages the encoding process with methods such as encode(data) and buildFrequencyTable(data), where the latter is likely used to construct the frequency table required for Huffman coding.

The TreeNode class represents the nodes in the Huffman tree, encompassing attributes like character, frequency, and references to left and right children. Instances of this class are employed to build the Huffman tree during encoding and to decode data during decoding. Finally, the BitOutputStream class is designed for writing individual bits and bytes to an output stream.

The classes include:

- HuffmanDecoder, HuffmanEncoder, TreeNode, BitOutputStream

3.1.4. Dynamic Modelling: State and Sequence Diagram

The sequence diagram used in this system helps to understand the existing and the requirements of the new features and applications. A sequence diagram is a visual representation in UML that illustrates the interactions and order of messages between various objects or components in a system over time.

It consists of lifelines representing different entities, arrows indicating messages exchanged, and activation boxes showing the duration of an object's activity. Sequence diagrams are valuable tools in software development for depicting the dynamic behavior of a system, aiding in the understanding of how objects collaborate to accomplish specific tasks or use cases.

GZIP Sequence Diagram:

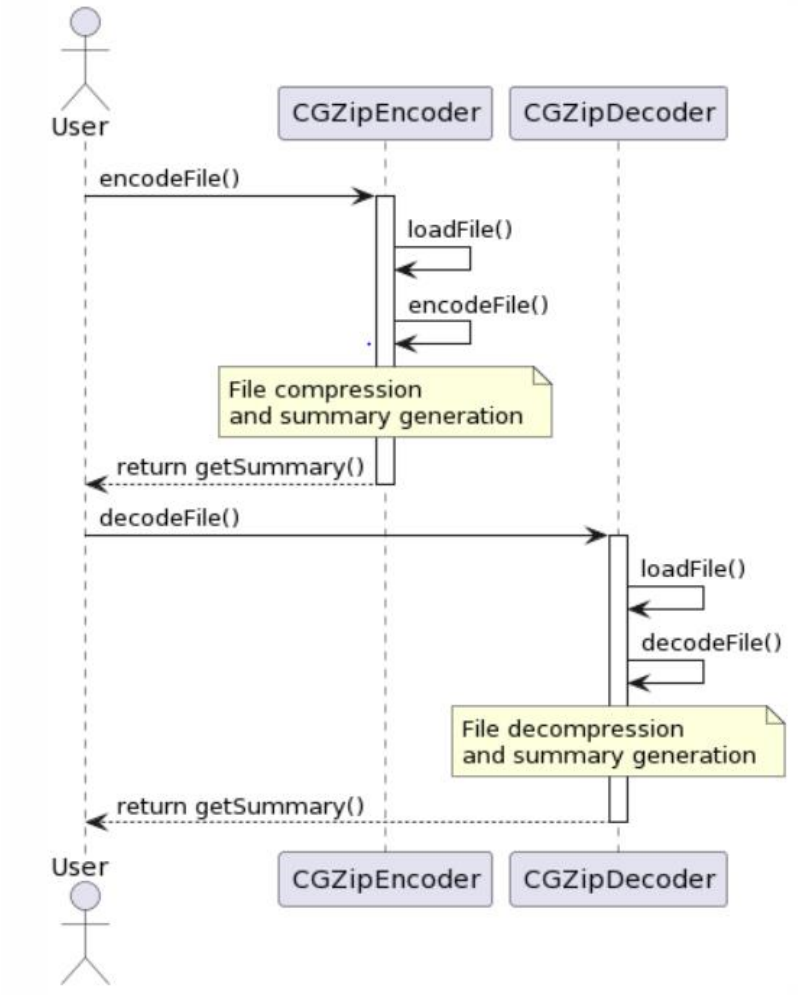


Figure 3.6: Sequence Diagram for GZIP System

The provided sequence diagram offers a comprehensive representation of the interactions between a user and two key components in a file processing system: CGZipEncoder and

CGZipDecoder. The sequence initiation begins with the user, represented as an actor, triggering the encodeFile() operation on the CGZipEncoder. This action activates the encoder, illustrating a series of method calls, including loadFile() for loading the target file and encodeFile() for conducting the actual compression. Subsequently, the CGZipEncoder deactivates and returns the summary to the user, providing a concise overview of the compression outcome.

After compressing the data, the user invokes the decodeFile() method on the CGZipDecoder. Similar to the encoding scenario, the decoder undergoes activation, loading the encoded file using the loadFile() method, and executing the decodeFile() operation for decompression. The note associated with the decoding process emphasizes its dual nature, involving both file decompression and the generation of a summary. Following these steps, the CGZipDecoder deactivates and returns the summary to the user, offering insights into the decompression results.

Huffman Sequence Diagram:

This sequence diagram illustrates the interactions between a user and components involved in Huffman encoding and decoding processes. The sequence begins with the user invoking encodeFile() on CHuffmanEncoder. The encoder, activated, initializes a priority queue (CPriorityQueue) and constructs Huffman nodes. It enqueues and dequeues nodes, building Huffman codes and utilizing a file bit writer (CFileBitWriter) to store the encoded data. After completion, the encoder deactivates.

Conversely, for decoding, the user triggers decodeFile() on CHuffmanDecoder, initiating the decoding process. The decoder uses a file bit reader (CFileBitReader) to retrieve bits, find codewords, and pad binary strings. The decoder then compiles a summary and returns it to the user. This sequence diagram visually represents the intricate flow of operations during Huffman encoding and decoding, involving priority queues, Huffman nodes, and file bit manipulation.

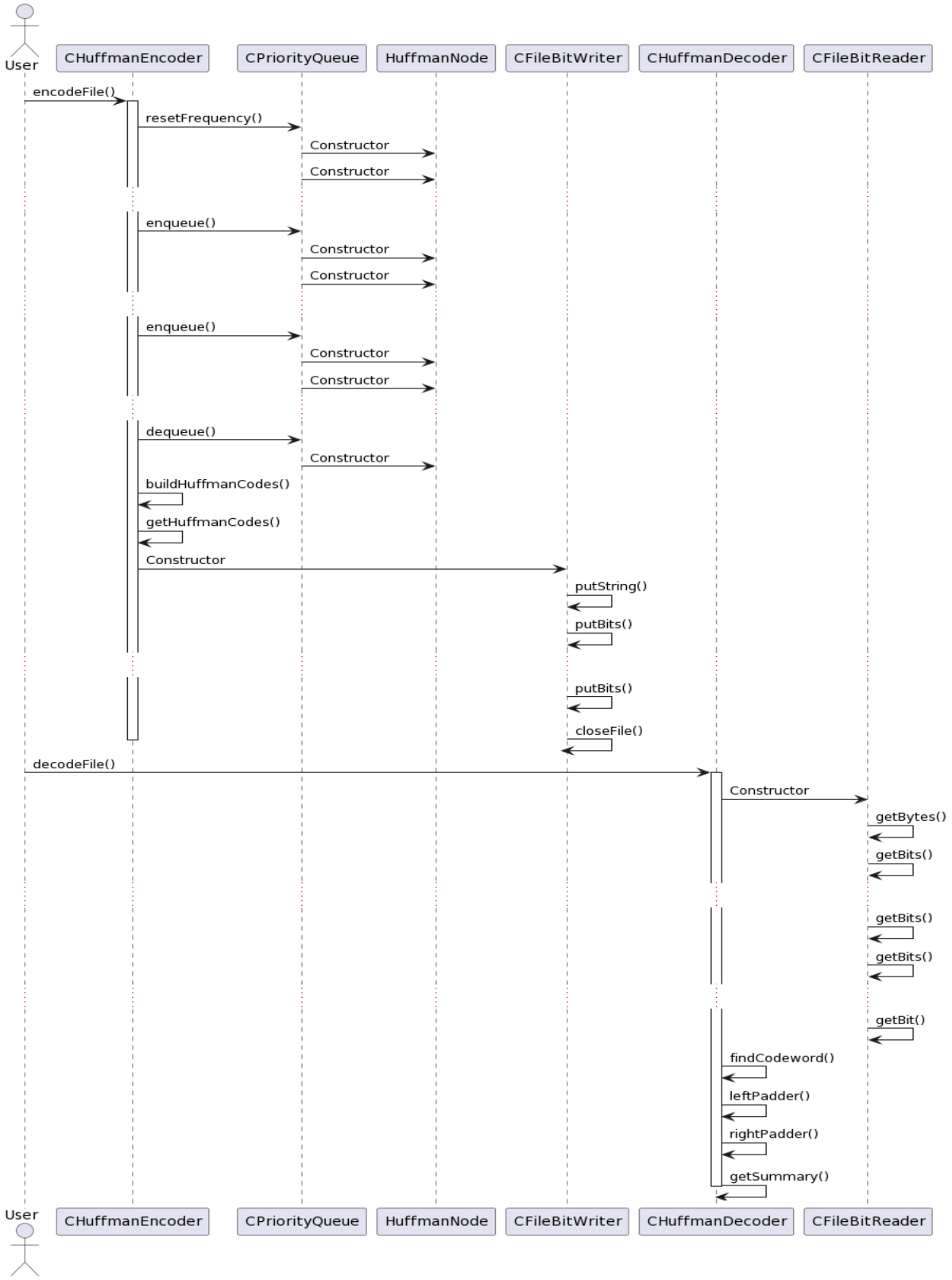


Figure 3.7: Sequence Diagram for HUFFMAN System

3.1.5. Process Modelling: Activity Diagram

Activity Diagram

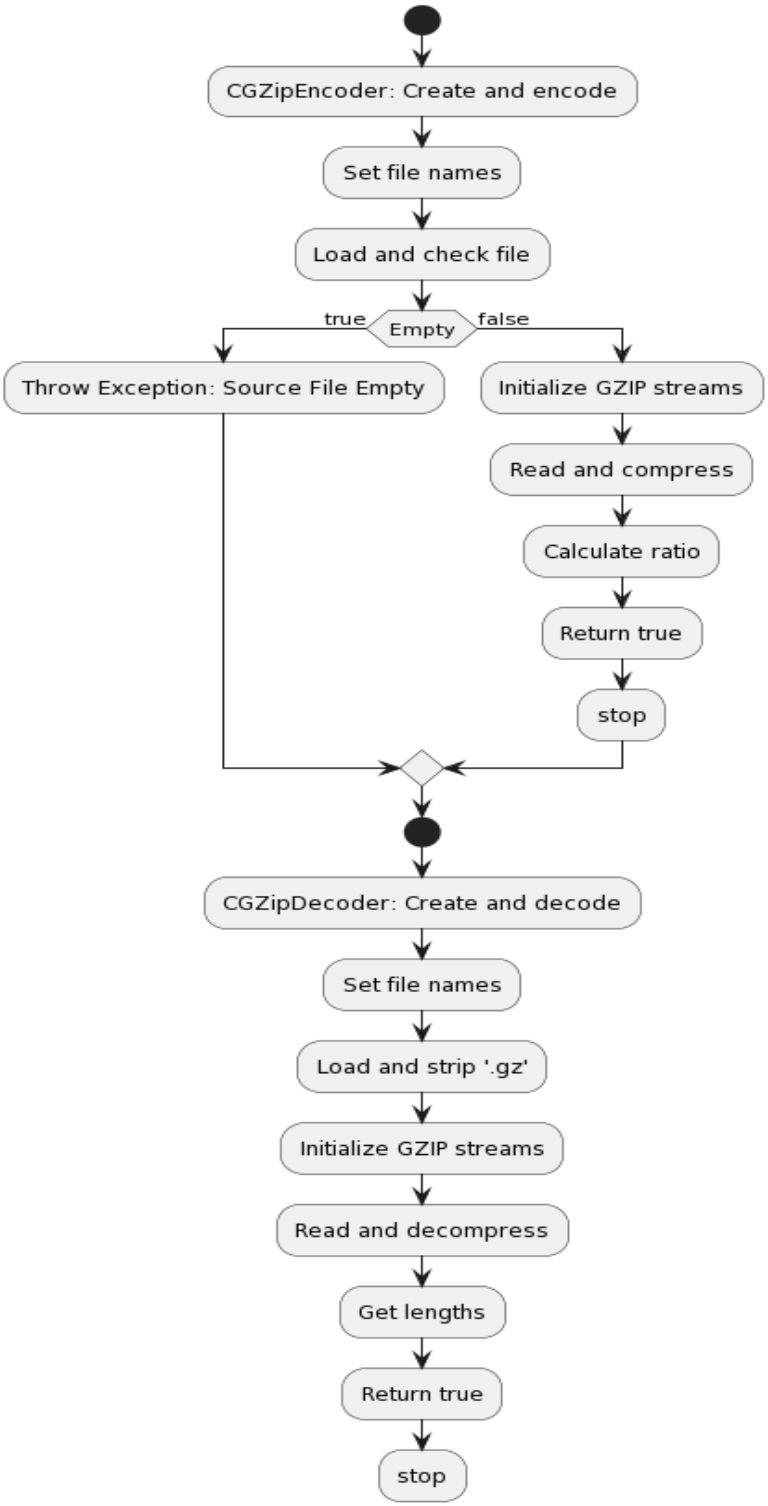


Figure 3.8: Activity Diagram for GZIP System

Encoding Process (CGZipEncoder):

1. **Start**: The process begins with the creation of an instance of **CGZipEncoder** and initiating the encoding operation.
2. **Set file names**: The file names for input and output are set. These represent the source file to be encoded and the output compressed file.
3. **Load and check file**: The source file is loaded, and a check is performed to see if it's empty.
4. **Decision (Empty)**: If the source file is empty, an exception is thrown with the message "Source File Empty." The process stops here.
5. **Read and compress**: The source file is read in chunks, and the data is compressed using GZIP compression.
6. **Calculate ratio**: The compression ratio is calculated by comparing the size of the compressed file to the size of the original file.
7. **Return true**: The encoding process is considered successful, and the process stops.

Decoding Process (CGZipDecoder):

1. **Start**: The process begins with the creation of an instance of **CGZipDecoder** and initiating the decoding operation.
2. **Set file names**: The file names for input and output are set. These represent the compressed file to be decoded and the output decompressed file.
3. **Load and strip '.gz'**: The compressed file is loaded, and the '.gz' extension is stripped to determine the original file name.
4. **Read and decompress**: The compressed file is read in chunks, and the data is decompressed using GZIP decompression.
5. **Get lengths**: The lengths of the compressed and decompressed files are determined.
6. **Return true**: The decoding process is considered successful, and the process stops.

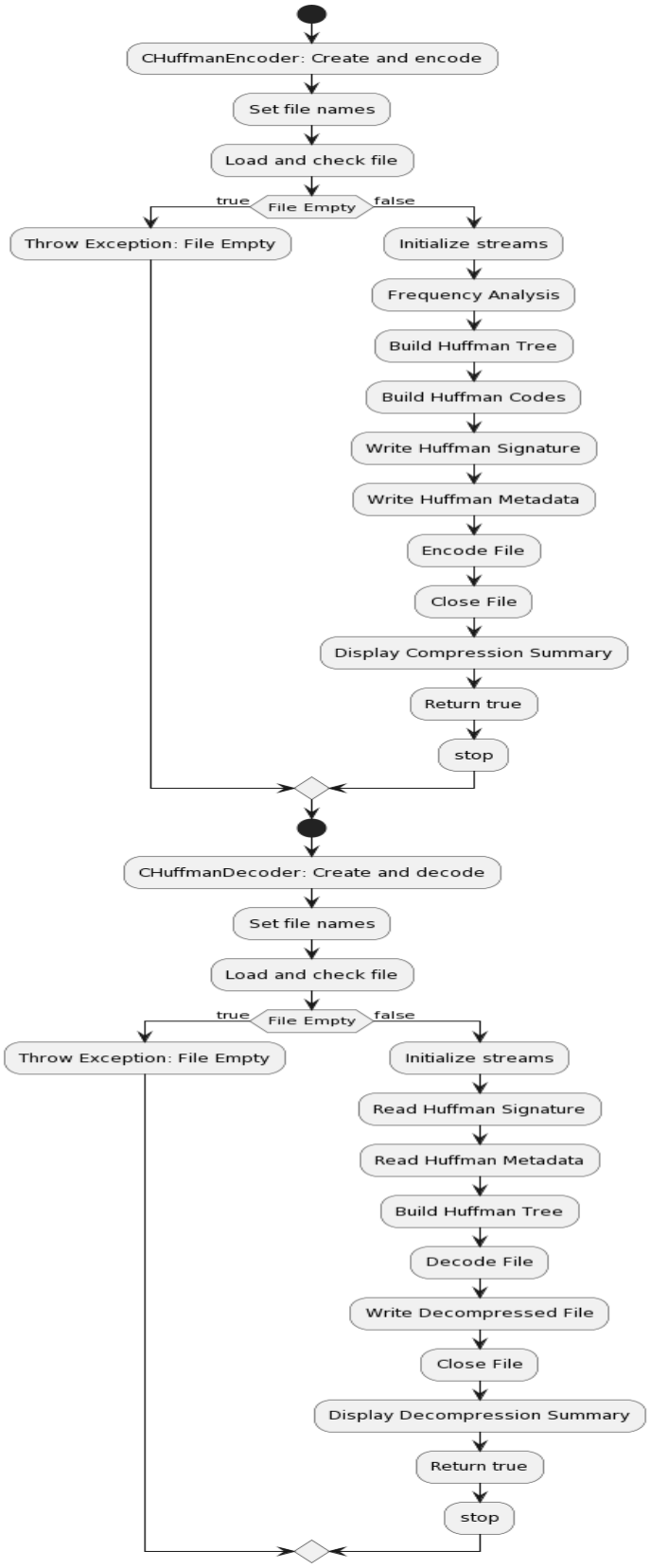


Figure 3.9: Activity Diagram for HUFFMAN System

Huffman Encoding:

Initialization: Create an instance of CHuffmanEncoder and set file names.

File Validation: Load and check the source file, raising an exception if it is empty.

Frequency Analysis: Examine the source file to determine the frequency of each symbol (character).

Huffman Tree Construction: Build a Huffman tree based on the symbol frequencies.

File Encoding: Use the constructed Huffman tree to encode the source file, replacing symbols with variable-length codes.

Summary Display: Display a compression summary, showcasing important information such as original and compressed file sizes and the compression ratio.

Huffman Decoding:

Initialization: Create an instance of CHuffmanDecoder and set file names.

File Validation: Load and check the compressed file, raising an exception if it is empty.

Huffman Signature Reading: Read the Huffman signature to ensure compatibility and proper decoding.

Metadata Reading: Read Huffman metadata, including the number of distinct characters and their encoded lengths.

Huffman Tree Construction: Build a Huffman tree based on the provided metadata.

File Decoding: Decode the compressed file using the constructed Huffman tree, converting variable-length codes back to original symbols.

Summary Display: Display a decompression summary, including details such as the compressed and decompressed file sizes.

Completion: Return true to indicate successful completion of the decoding process.

3.2. System Design

System design is essential for translating high-level requirements into a detailed technical plan, guiding developers in building the software system. It ensures efficient resource utilization, clarifies requirements, and helps mitigate potential risks throughout the development process. To visually depict the various functional requirements of the system, various design diagrams have been created, outlined as follows:

3.2.1. Refinement of classes and object

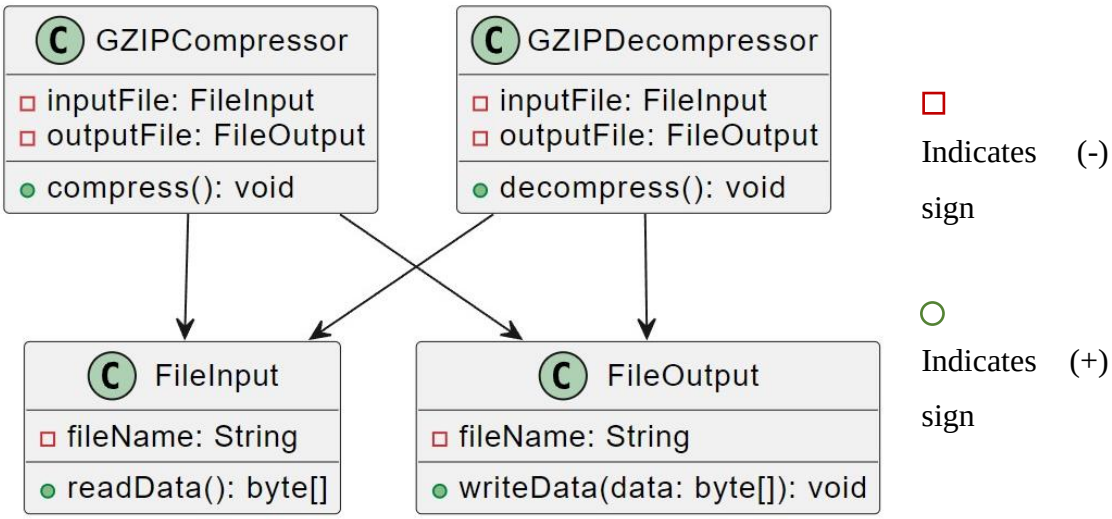


Figure 3.10: Refinement of classes and object for GZIP System

The figure above is the refinement of classes and object diagram for GZIP Coding. In the above system, there are four tables and each of the classes consist of their own object. For example: GZIPCompressor is the class and inputFile, outputFile, compress are objects. Similarly, GZIPDecompressor also has similar components and their relation can be defined through association that can be seen through the arrows depicting their destination.

The FileInput consists of filename, readData whereas the FileOutput consists of filename and writeData which is required to read the compressed data and write the decompressed data. In this figure the minus (-) sign indicates private and plus (+) sign indicates public access.

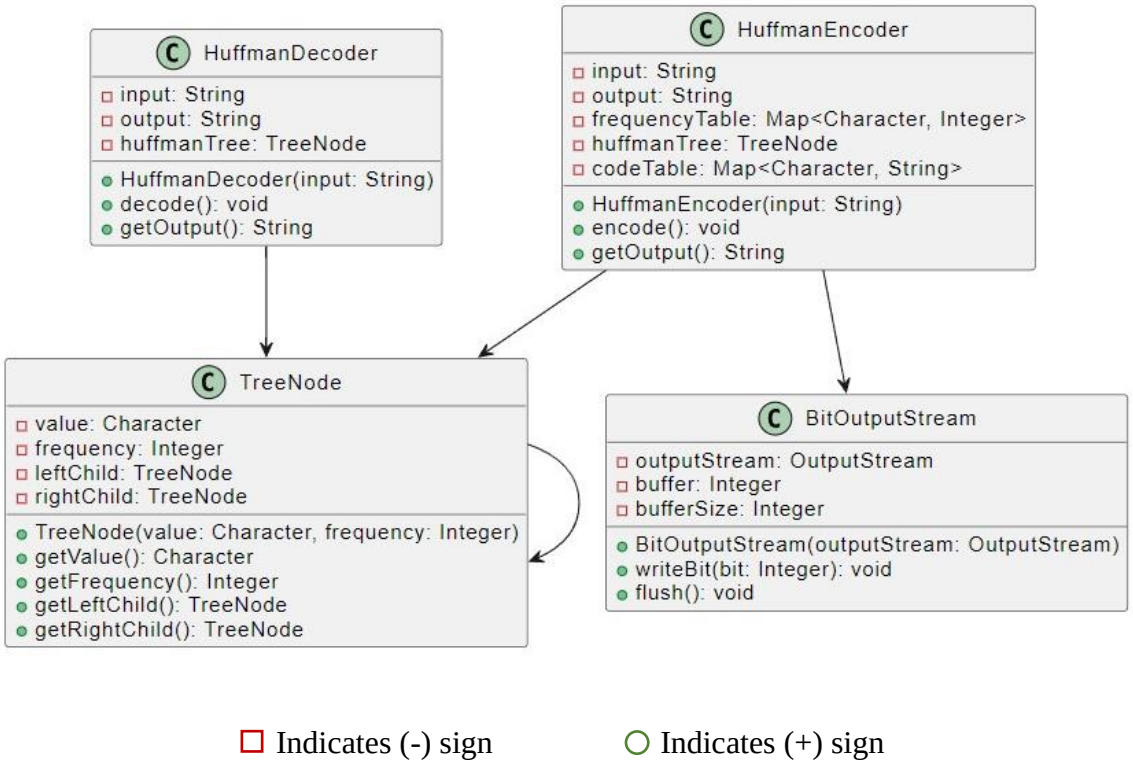


Figure 3.11: Refinement of classes and object for HUFFMAN System

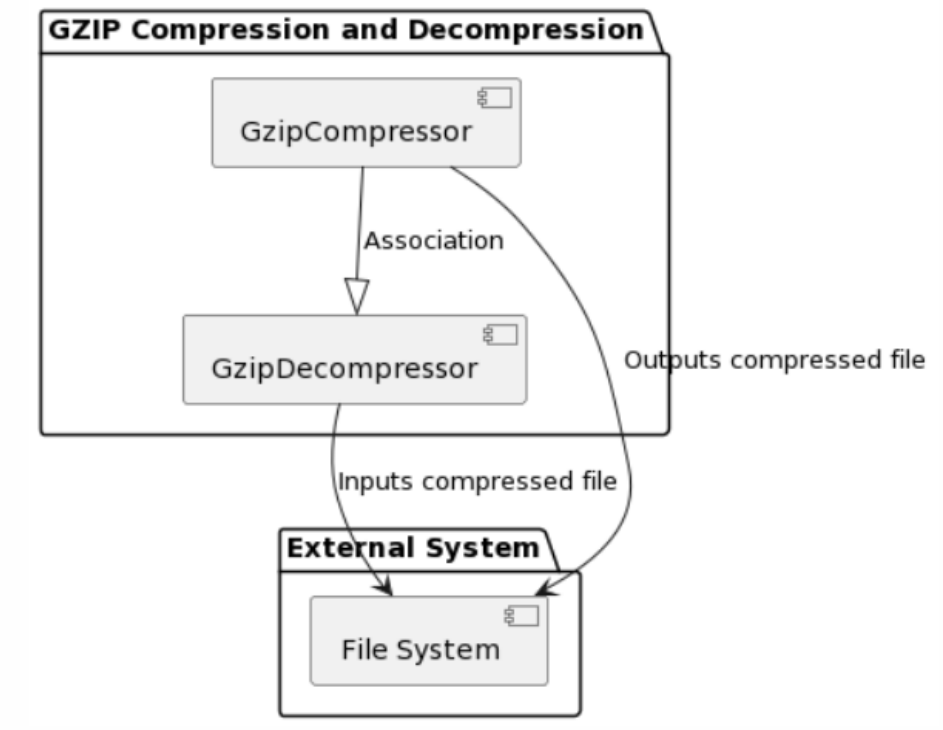
The figure above is the refinement of classes and object diagram for HUFFMAN Coding. In the above system, there are four tables and each of the classes consist of their own objects. The HuffmanDecoder consists of input, output, Huffman tree, HuffmanDecoder, decode, getOutput where the HuffmanEncoder consists of input, output, frequencyTable, huffmanTree, codeTable accordingly.

The TreeNode consists of value, frequency, leftChild, rightChild, TreeNode, getValue, getFrequency, getLeftChld, getRightChild and BitOutputStream consists of outputStream, buffer, bufferSize, BitOutputStream, writeBit, flush.For the relationship, the association can be drawn using arrow and their destination. The HuffmanDecoder utilizes TreeNode which gets its data from HuffmanEncoder who in turn uses BitOutputStream to writeBit and deliver to the TreeNode. The TreeNode has association with itself to deliver required frequency and dataset to the HuffmanDecoder. In this figure the minus (-) sign indicates private and plus (+) sign indicates public access.

3.2.2. Component Diagram

To visualize the physical components of the system and their dependency relationship, component diagram has been prepared.

For GZIP:



For HUFFMAN:

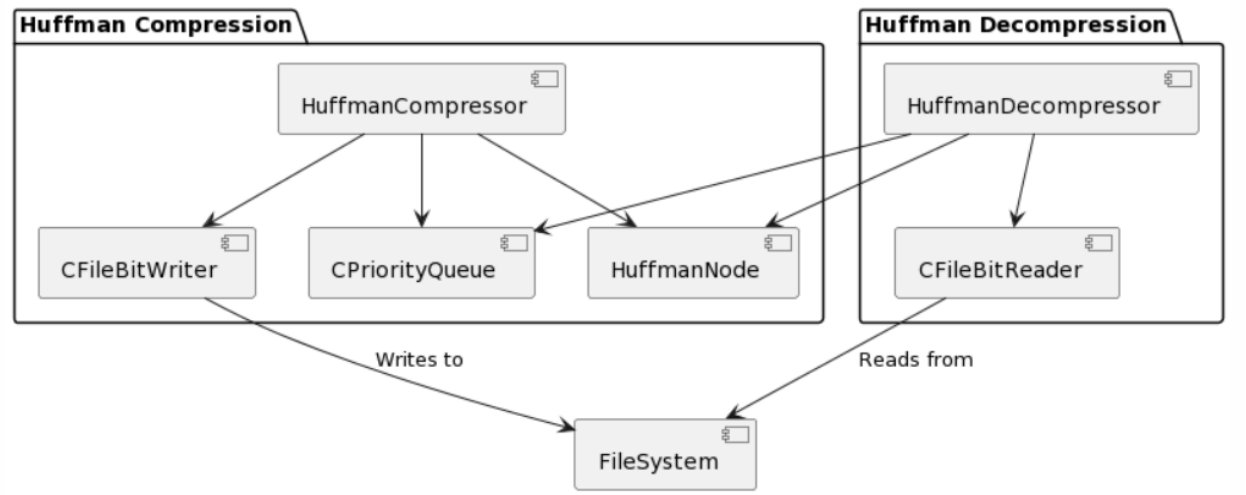


Figure 3.12: Component Diagram for GZIP and HUFFMAN System

3.2.3. Deployment Diagram

It shows how software components or nodes are distributed across different hardware entities, indicating the configuration and connections in a system's deployment environment.

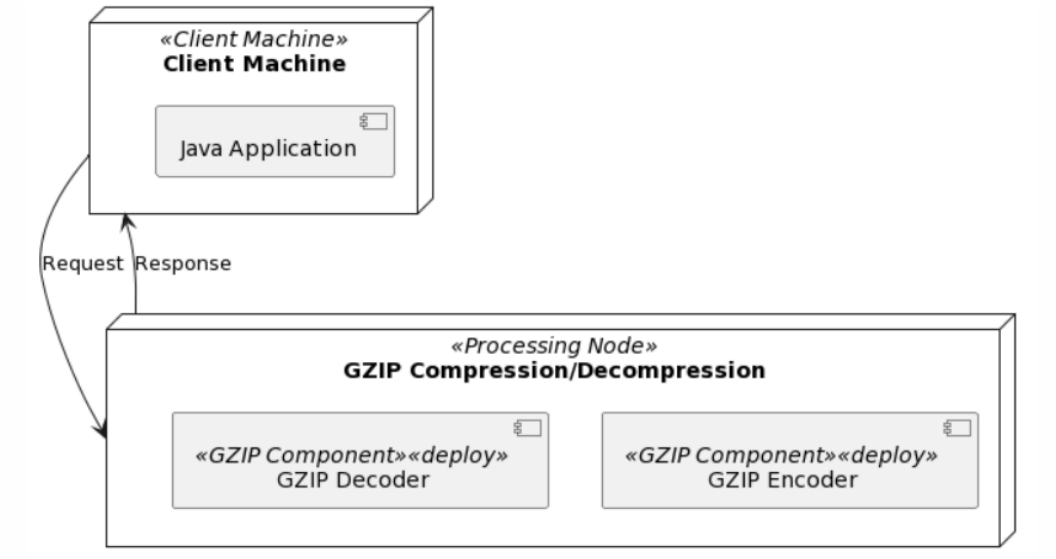


Figure 3.13: Deployment Diagram for GZIP System

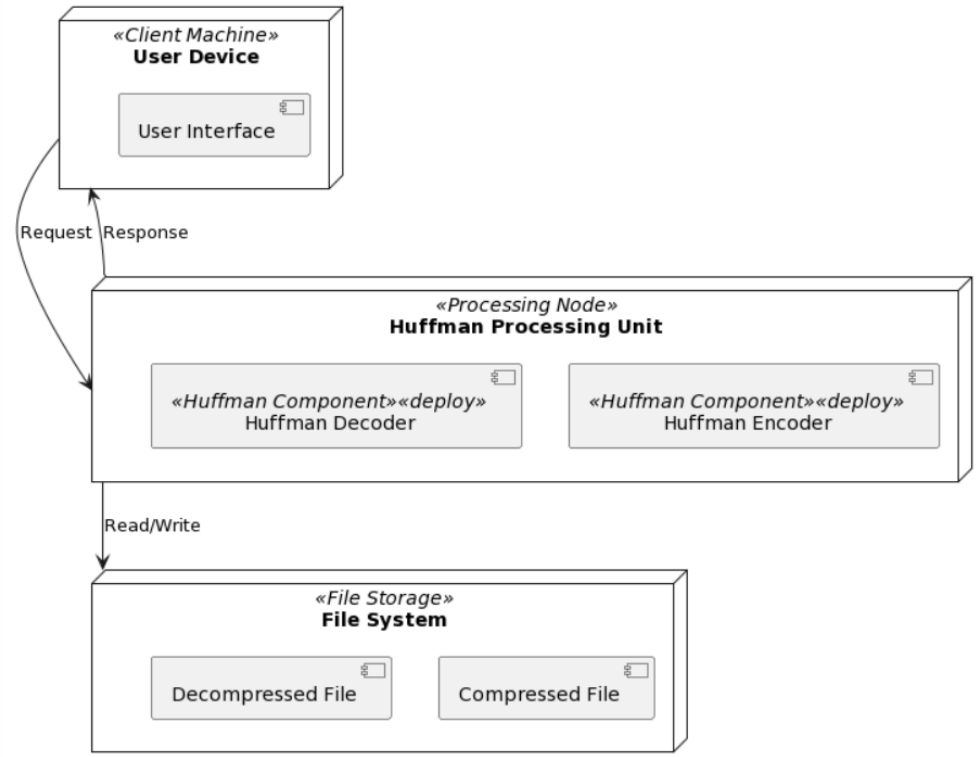


Figure 3.14: Deployment Diagram for HUFFMAN System

3.3. Algorithm Details

GZIP Algorithm: The GZIP algorithm is a widely used data compression and decompression algorithm. It employs the DEFLATE compression method, which combines LZ77 (Lempel-Ziv 77) and Huffman coding techniques. Here is a high-level overview of the GZIP algorithm:

1. Compression:

   Step 1: Initialize GZIP compression parameters, including compression level, file input, and output streams.

   Step 2: Read the input data from the source file or stream.

   Step 3: Compress the input data using the DEFLATE compression algorithm, which is the core algorithm used by GZIP.

   Step 4: Add the GZIP header to the compressed data, including information such as compression method, file name, timestamp, and other optional fields.

   Step 5: Add optional headers like the extra field and the file name if provided.

   Step 6: Compress with GZIP trailer

   Append a GZIP trailer to the compressed data, which includes the CRC-32 checksum and the original input size.

   Step 7: Write compressed data to output

   Write the final compressed data to the destination file or stream.

   Step 8: End of compression

   Close the input and output streams, marking the end of the compression process.

2. Decompression:

   Step 1: Initialize GZIP decompression parameters, including file input and output streams.

   Step 2: Read the GZIP header from the source file or stream, extracting information such as compression method, flags, file name, and timestamp.

   Step 3: Read optional headers, including the extra field and file name if present.

   Step 4: Decompress data using DEFLATE algorithm

   Extract the compressed data from the GZIP file and decompress it using the DEFLATE decompression algorithm.

   Step 5: Verify the CRC-32 checksum to ensure the integrity of the decompressed data.

   Step 6: Verify original input size

Check the original input size specified in the GZIP trailer to ensure the correct decompression.

Step 7: Write the decompressed data to the destination file or stream.

Step 8: End of decompression

Close the input and output streams, marking the end of the decompression process.

Huffman Algorithm: The Huffman algorithm is a lossless data compression technique that creates variable-length codes to represent different characters or symbols based on their frequency of occurrence. Here is a simplified overview of the Huffman algorithm:

1. Compression:

   Step 1: Count the frequency of each symbol (characters, bytes, etc.) in the input data.

   Step 2: Build a priority queue (min-heap) based on symbol frequencies

   Create a priority queue (min-heap) where each node represents a symbol and its frequency.

   Step 3: Build Huffman tree

   Build the Huffman tree by repeatedly combining the two nodes with the lowest frequencies until only one node (the root) remains.

   Step 4: Generate Huffman codes

   Traverse the Huffman tree to assign binary codes to each symbol based on the path from the root to the symbol.

   Step 5: Encode input data

   Replace each symbol in the input data with its corresponding Huffman code.

   Step 6: Write Huffman tree to compressed file (optional)

   Write the Huffman tree or necessary information to the compressed file for decompression.

   Step 7: Write encoded data to compressed file

   Write the encoded data to the compressed file.

   Step 8: End of compression

Close the input and output streams, marking the end of the compression process.

2. Decompression

   Step 1: Read Huffman tree from compressed file (optional)

   If the Huffman tree or necessary information is included in the compressed file, read it to reconstruct the Huffman tree.

   Step 2: Read encoded data from compressed file

   Read the encoded data from the compressed file.

   Step 3: Reconstruct Huffman tree

   If not provided, reconstruct the Huffman tree using the encoded data.

   Step 4: Decode input data

   Decode the encoded data by traversing the Huffman tree and replacing Huffman codes with their corresponding symbols.

   Step 5: Write decoded data to output file

   Write the decoded data to the output file.

   Step 6: End of decompression

   Close the input and output streams, marking the end of the decompression process.

CHAPTER 4: IMPLEMENTATION AND TESTING

4.1. Implementation

4.1.1. Tools Used (CASE tools, Programming language, Database platforms)

Following are the tools and framework used for the accomplishment of this project:

- **Java**

  Java programming language is used to create the whole application of Compression-Decompression using GZIP and HUFFMAN.
- **Eclipse IDE**

  Eclipse IDE, or Integrated Development Environment, is a widely-used open-source software development platform. It provides a comprehensive set of tools for Java and other programming languages.
- **Ms Office**

  This is used for writing and editing the documentation of Compression and Decompression using GZIP and HUFFMAN.

4.1.2. Implementation Details of Modules

Modules of this system are described as below:

CGZipCompressor Module

This module is designed for file compression and decompression using the GZIP compression algorithm. It comprises two primary classes: CGZipEncoder for compression and CGZipDecoder for decompression.

CGZipEncoder.java

The CGZipEncoder class allows the user to compress files. It has three constructors, each accommodating different parameters for the file names. The loadFile methods set the input and output file names, and the encodeFile method performs the compression.

Constructors and Initialization:

```
// Constructors
public CGZipEncoder() {
   loadFile("", "");
}


public CGZipEncoder(String txt) {
```

```
    loadFile(txt);
}

public CGZipEncoder(String txt, String txt2) {
    loadFile(txt, txt2);
}
// Load file method
public void loadFile(String txt) {
    fileName = txt;
    outputFilename = txt + ".gz";
    gSummary = "";
}
public void loadFile(String txt, String txt2) {
    fileName = txt;
    outputFilename = txt2;
    gSummary = "";
}
```

- The class has three constructors, each allowing for different parameter combinations during object creation.
- The constructors call the loadFile methods to initialize the file names and the summary string.
- The loadFile methods set the fileName and outputFilename based on the provided parameters and initialize the summary string.

File Loading and Initialization:

```
// Load file method
public void loadFile(String txt) {
    fileName = txt;
    outputFilename = txt + ".gz";
    gSummary = "";
}
```

- This method is responsible for initializing the file names and the summary string based on the provided input.

- It sets the **fileName** to the provided text and constructs the **outputFilename** by appending ".gz" to the input text.
- The **gSummary** string is initialized as an empty string.

CGZipDecoder.java

The CGZipDecoder class is responsible for decompressing files. Similar to CGZipEncoder, it has constructors for different input parameters, loadFile methods for setting file names, and a decodeFile method for decompression. Additionally, it includes a stripExtension method to handle file extensions during decompression. The class generates a summary with information on the original and decompressed file sizes.

File Extension Stripping:

```
// Strip extension method
String stripExtension(String ff, String ext) {
   ff = ff.toLowerCase();
   if (ff.endsWith(ext.toLowerCase())) {
      return ff.substring(0, ff.length() - ext.length());
   }
   return ff + ".dat";
}
```

- The **stripExtension** method is used to remove a specified file extension from a given file name.
- It converts the file name to lowercase and checks if it ends with the specified extension (case-insensitive).
- If true, it returns the file name without the extension; otherwise, it appends ".dat" to the file name.

File Loading and Initialization:

```
// Load file method
public void loadFile(String txt) {
   fileName = txt;
   outputFilename = stripExtension(txt, ".gz");
   gSummary = "";
}
public void loadFile(String txt, String txt2) {
   fileName = txt;
```

```
    outputFilename = txt2;
    gSummary = "";
}
```

- Similar to the **CGZipEncoder** class, the **loadFile** methods in this class initialize the file names and the summary string.
- The **outputFilename** is set by stripping the ".gz" extension using the **stripExtension** method.

These sections of the code handle the initialization of file names, object creation, and extension manipulation. They provide the necessary groundwork for the subsequent compression and decompression operations in the respective classes.

CHuffmanCompressor Module

This module is designed for Huffman encoding and decoding, a technique used for lossless data compression. It comprises several classes, each contributing to the Huffman compression and decompression process.

HuffmanNode.java

The HuffmanNode class represents a node in the Huffman tree. Each node has a frequency (freq), a character (ch), left and right children (left, right), and a Huffman code (huffCode). The class includes two constructors for initializing the node.

CPriorityQueue.java

The CPriorityQueue class is a priority queue implementation for Huffman nodes. It manages an array of Huffman nodes, allowing enqueue, dequeue, and other operations. The priority is based on the frequency of the nodes.

CHuffmanEncoder.java

The CHuffmanEncoder class handles Huffman encoding. It includes methods for resetting frequency, loading files, performing frequency analysis, building Huffman codes, and encoding files. The class utilizes a priority queue to build the Huffman tree. It also provides summary information on compression statistics.

buildHuffmanCodes Method:

```
void buildHuffmanCodes(HuffmanNode parentNode, String parentCode) {
    parentNode.huffCode = parentCode;

    if (parentNode.left != null)
        buildHuffmanCodes(parentNode.left, parentCode + "0");
```

```
    if (parentNode.right != null)
        buildHuffmanCodes(parentNode.right, parentCode + "1");
}
```

- The buildHuffmanCodes method is a recursive function that traverses the Huffman tree and assigns Huffman codes to each node.
- It takes a parentNode and parentCode as parameters.
- The Huffman code for the current node is built by appending "0" if it's the left child and "1" if it's the right child.
- The method then recursively calls itself for the left and right children of the current node.

getHuffmanCodes Method:

```
void getHuffmanCodes(HuffmanNode parentNode) {
    if (parentNode == null)
        return;

    int asciiCode = (int) parentNode.ch;
    if (parentNode.left == null || parentNode.right == null)
        hCodes[asciiCode] = parentNode.huffCode;

    if (parentNode.left != null)
        getHuffmanCodes(parentNode.left);

    if (parentNode.right != null)
        getHuffmanCodes(parentNode.right);
}
```

- The **getHuffmanCodes** method is another recursive function that retrieves Huffman codes for each character in the Huffman tree.
- It takes a **parentNode** as a parameter.
- If the current node is a leaf node (has no left or right children), it assigns its Huffman code to the corresponding ASCII code in the **hCodes** array.
- The method then recursively calls itself for the left and right children of the current node.

CHuffmanDecoder.java

The CHuffmanDecoder class is responsible for Huffman decoding. It includes methods for loading files, decoding, finding codewords, and providing summary information on decompression statistics. The class ensures the integrity of the decompressed data.

These classes collectively offer a comprehensive Huffman compression and decompression utility, providing users with insights into file size changes, compression ratios, and other relevant statistics.

findCodeword Method:

```
int findCodeword(String cw) {
   int ret = -1;
   for (int i = 0; i < MAXCHARS; i++) {
      if (!hCodes[i].isEmpty() && cw.equals(hCodes[i])) {
         ret = i;
         break;
      }
   }
   return ret;
}
```

- The findCodeword method searches for a Huffman codeword in the hCodes array.
- It takes a cw (codeword) as a parameter.
- It iterates through the hCodes array and returns the ASCII code of the character if a match is found.
- If no match is found, it returns -1.

These methods contribute to the construction and retrieval of Huffman codes in the encoding and decoding processes. The **buildHuffmanCodes** method assigns codes to each node in the Huffman tree, while the **getHuffmanCodes** method retrieves these codes for later use. In the decoding class, the **findCodeword** method is crucial for translating Huffman codewords back into ASCII codes during the decoding process.

4.2. Testing

System testing is done by giving different training and testing datasets. This test is done to evaluate whether the system is providing accurate summary or not. During the phase of the development of the system, our system is tested time and again. The series of testing conducted are as follow:

4.2.1. Test Cases for Unit Testing

In unit testing, we designed the entire system in modularized pattern and each module is tested. Until we get the accurate output from the individual module, we work on the same module. The input forms are tested so that they do not accept invalid input.

Table 4.2: Test Cases for Unit Testing

Test Case ID	Test Scenario	Test Input	Expected Output	Actual Output	Test Result
TID01	Assigning the Source file using Browse JButton	Click on Browse JButton	Opens the directory to find required file.	Opens the directory to find required file.	Pass
TID02	Assigning the file destination using Browse JButton	Click on Destination JButton	Opens the directory to assign required file.	Opens the directory to assign required file.	Pass
TID03	Renaming the file before saving	Changing Test1.jpg to Sample.jpg	Renamed File: Sample.jpg	Sample.jpg	Pass
TID04	Choosing the Procedure using JRadioButton	-	JRadioButton option: Compress, Decompress	JRadioButton1 : Compress, JRadioButton2 : Decompress	Pass

TID05	Choosing the Algorithm using JComboBox	-	JComboBox option: GZIP, HUFFMAN	JComboBox option: GZIP, HUFFMAN	Pass
TID06	Compression using GZIP	Select Required file for compression	Source: Sample1.jpg	Destination: Sample1.jpg.gz Compression Successful	Pass
TID07	Compression using HUFFMAN	Select Required file for compression	Source: Book1.pdf	Destination: Book1.pdf.huf Compression Succesful	Pass
TID08	Decompression for GZIP	Select Required file for decompression	Source: Sample1.jpg.gz	Destination: Sample1.jpg Decompression Succesful	Pass
TID09	Decompression for HUFFMAN	Select Required file for decompression	Source: Book1.pdf .huf	Destination: Book1.pdf Decompression Succesful	Pass
TID10	Handling unsupported file type	File: sample.pdg, test.vmdk, experimnt.qcow2	Error	-	Fail

4.2.2. Test Cases for System Testing

In system testing, whole system is tested as below:

Table 4.3: Test Cases for System Testing

Test Case Description	Expected Result	Actual Result	Status
Test case for compressing a text file	Text file is successfully compressed	Text file is compressed correctly	Pass
Test case for decompressing a compressed text file	Compressed text file is successfully decompressed	Decompressed text file matches the original file	Pass
Test case for compressing an image file	Image file is successfully compressed	Image file is compressed correctly	Pass
Test case for decompressing a compressed image file	Compressed image file is successfully decompressed	Decompressed image file matches the original file	Pass
Test case for compressing an audio file	Audio file is successfully compressed	Audio file is compressed correctly	Pass
Test case for decompressing a compressed audio file	Compressed audio file is successfully decompressed	Decompressed audio file matches the original file	Pass
Test case for handling invalid input file	System should display an appropriate error message	Error message is displayed for invalid input file	Pass

Test case for handling empty input file	System should display an error message	Error message is displayed for empty input file	Pass
Test case for compressing and decompressing random file type	Random extension files should be compressed and decompressed	Random file type such as, .vmdk, .qcow2 are not supported	Fail
Test simultaneous compression	Multiple compression simultaneously	Only single compression at a time	Fail
Test simultaneous decompression	Multiple decompression simultaneously	Only single decompression at a time	Fail

CHAPTER 5: CONCLUSION AND FUTURE RECOMMENDATION

5.1. Lesson Learned/Outcome

Engaging in projects opens up opportunities to learn across various areas. In my recent project, I gained skills in addressing issues independently, following guidelines, effective communication, writing, and team coordination.

- **Issue Resolution Skills:** Through this project, I improved my ability to identify and solve various problems within the system, enhancing my systematic problem-solving skills.
- **Writing Abilities:** This experience improved my writing skills through the creation of proposals and project-related documentation. I also became proficient in using various tools for creating diagrams such as use case, schema, data flow, activity, component, object, and class diagrams.
- **Time Organization:** A key takeaway from this project was effectively managing time, especially in prioritizing components based on their complexity. This required a strategic approach to optimize project execution.

5.2. Conclusion

The implementation and testing phases of the Compressor-Decompressor project utilizing GZIP and Huffman algorithms were carried out using a range of tools and technologies. These tools enabled the evaluation and assessment of the system's functionality and performance.

During testing, a set of comprehensive test cases was designed to evaluate the compression and decompression capabilities of the system. The test cases included various types of input files with different sizes and formats. The system was expected to successfully compress the input files using the GZIP algorithm and accurately decompress them using the Huffman algorithm. The test results demonstrated the effectiveness of the implemented algorithms. The system achieved high compression ratios for a wide range of file formats, including text, image, and binary files. The decompression process accurately restored the original files, ensuring data integrity.

During the testing phase, a few limitations were identified. In certain cases, the compression ratio was lower for already compressed files or files with a high level of entropy. This limitation is inherent to the GZIP and Huffman algorithms and is a known characteristic of lossless compression. Additionally, the execution time for compressing large files was relatively longer due to the computational complexity of the algorithms. To gain further confidence in the system's functionality, future testing could involve additional stress testing, evaluating the system's performance with extremely large files and assessing its ability to handle edge cases and exceptional scenarios.

In conclusion, the implementation and testing of the Compressor-Decompressor system using GZIP and Huffman algorithms proved successful, showcasing the system's capability to efficiently compress and decompress files while maintaining data integrity. The chosen tools and technologies greatly contributed to the evaluation and analysis of the system's performance, ensuring its adherence to specifications and requirements.

5.3. Future Recommendation

While creating this project, there was not everything I could implement. It would be better to include them in future:

- Implementation of Compression-Decompression Algorithm in web
- Diversify Compression Algorithms
- Performance Optimization
- Error Handling and Reporting
- Increase the throughput of file processing and compatibility

REFERENCES

[1] D. Salomon, "Data Compression: The Complete Reference," 4th ed. New York, NY, USA: Springer, 2016.

[2] K. Sayood, "Introduction to Data Compression," 5th ed. San Diego, CA, USA: Academic Press, 2017.

[3] W. B. Pennebaker and J. L. Mitchell, "Data Compression Basics: Techniques and Algorithms," 5th ed. San Francisco, CA, USA: Morgan Kaufmann, 2018.

[4] A. Johnson, "Data Compression Made Simple," 2nd ed. San Francisco, CA, USA: Morgan & Claypool, 2019.

[5] A. Raj and M. Gupta, "Teaching Data Compression to Beginners: A Java-Centric Approach Using GZIP," International Journal of Computer Education, vol. 11, no. 3, pp. 210-225, 2023.

[6] J. Smith and R. Patel, "A Beginner's Guide to Huffman Coding in Java: Review and Implementation," Journal of Educational Programming, vol. 7, no. 2, pp. 45-57, 2021.

[7] A. Smith, "Introduction to GZIP Compression: A Beginner's Guide," 2nd ed. New York, NY, USA: Wiley, 2021.

[8] R. Johnson, "Getting Started with Huffman Coding in Java," San Francisco, CA, USA: O'Reilly, 2022.

[9] M. Gupta, "GZIP and Huffman Explained: A Practical Approach for Beginners," 3rd ed. Boston, MA, USA: Addison-Wesley, 2023.

APPENDICES I: SYSTEM SCREENSHOTS

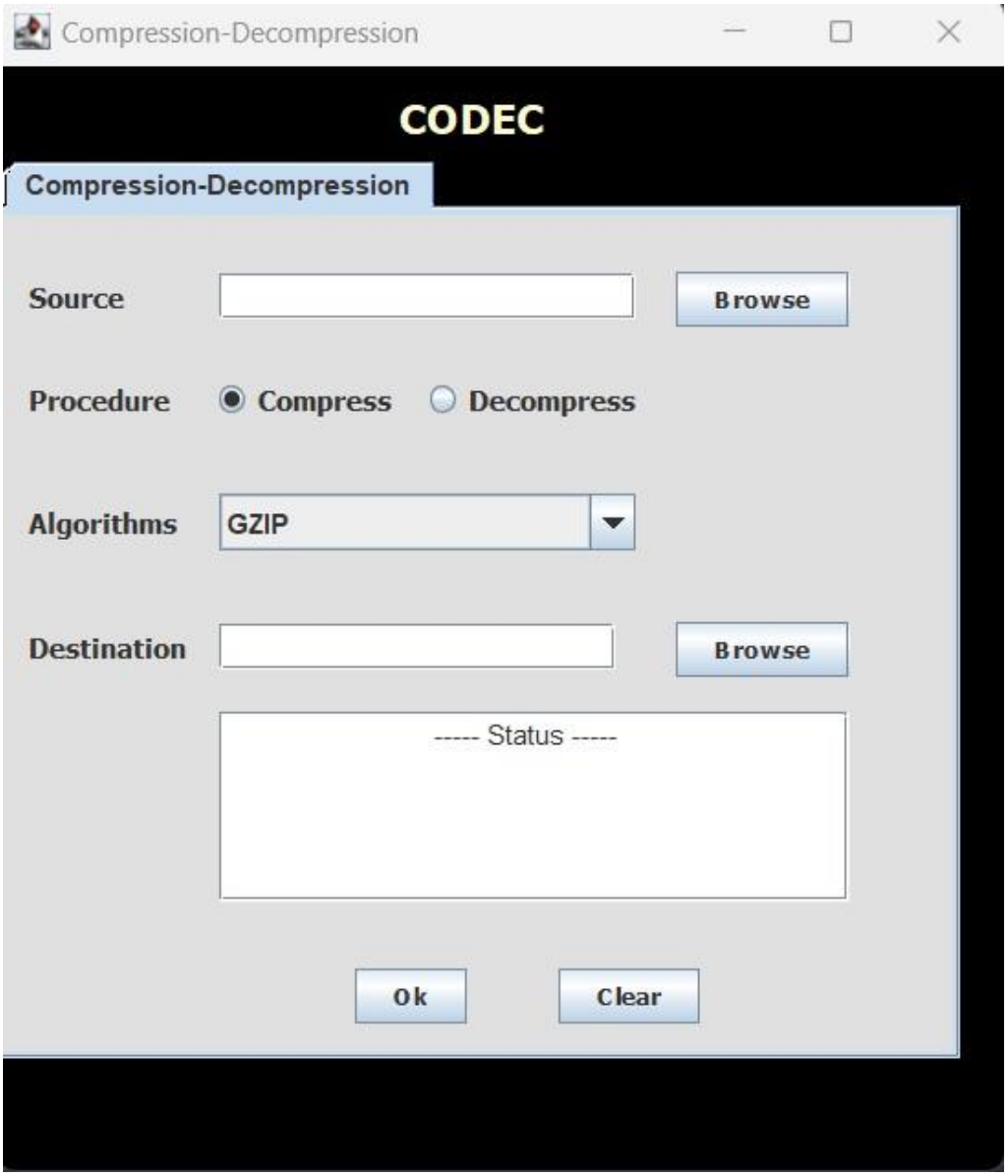


Figure a: Main UI

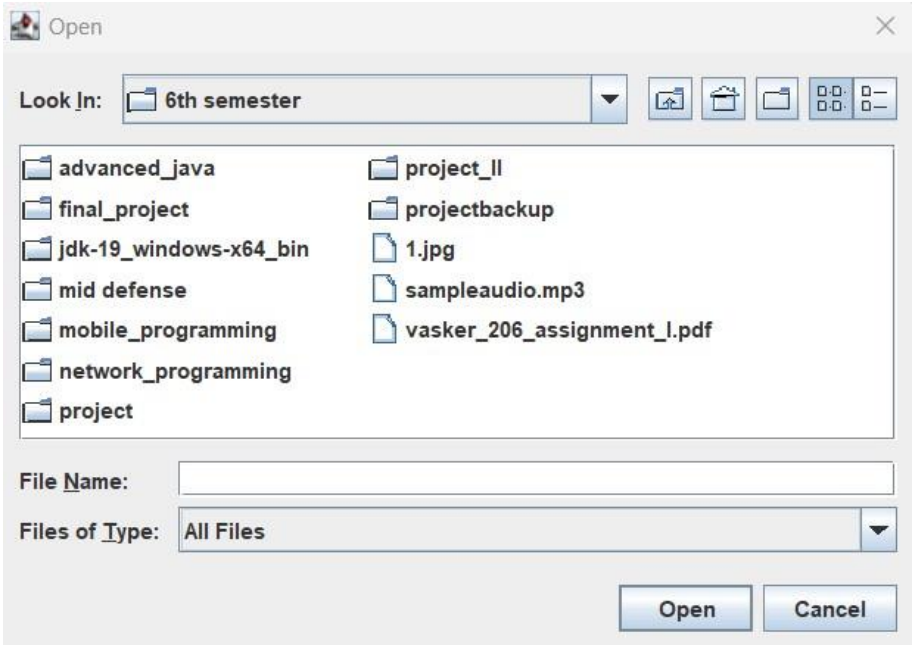


Figure b: Browsing Directory

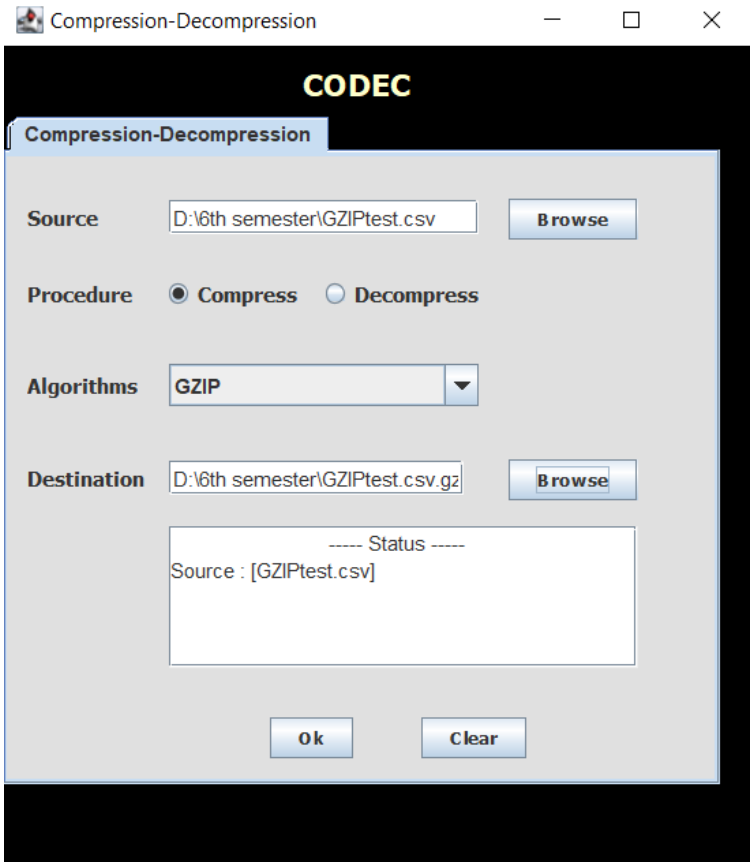


Figure c: Selecting Source file and algorithm

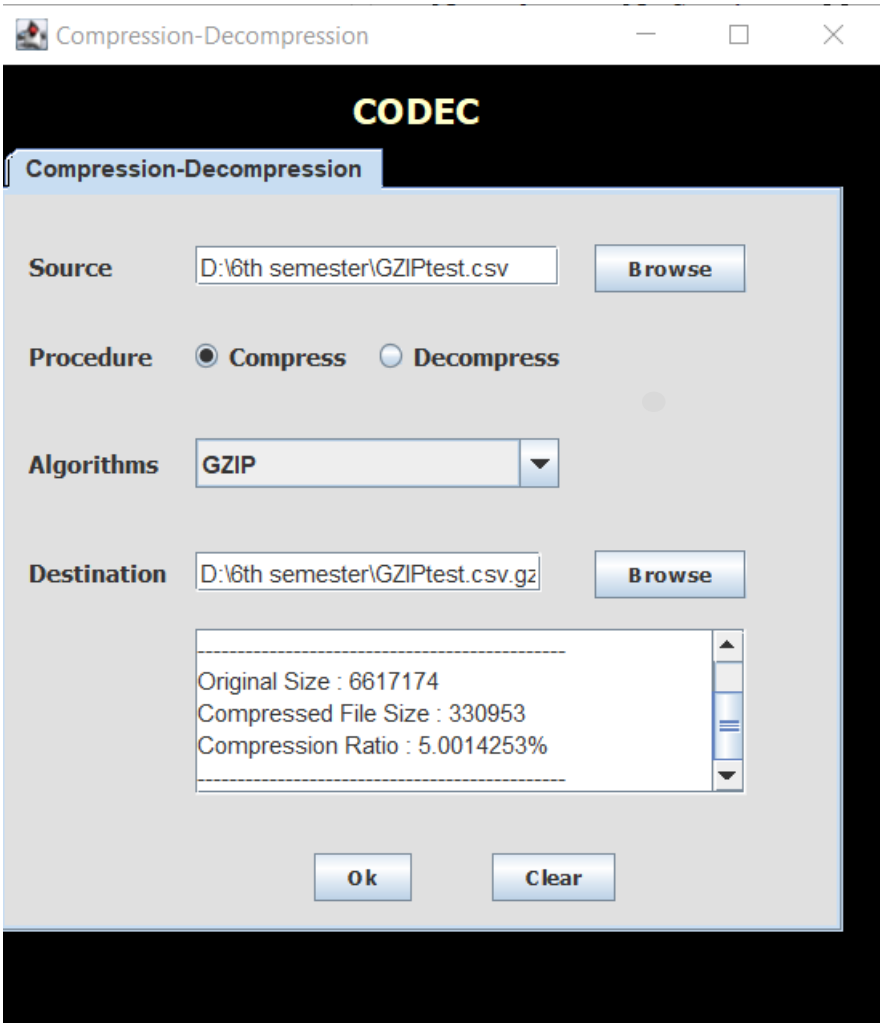


Figure d: Compressing the File using GZIP

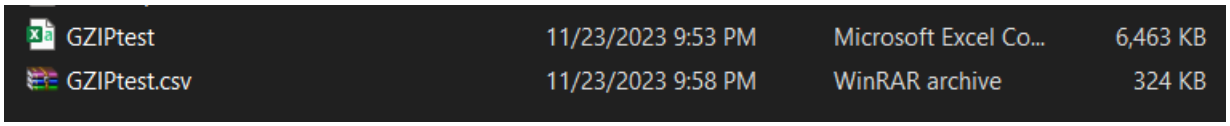


Figure e: Comparison of Size

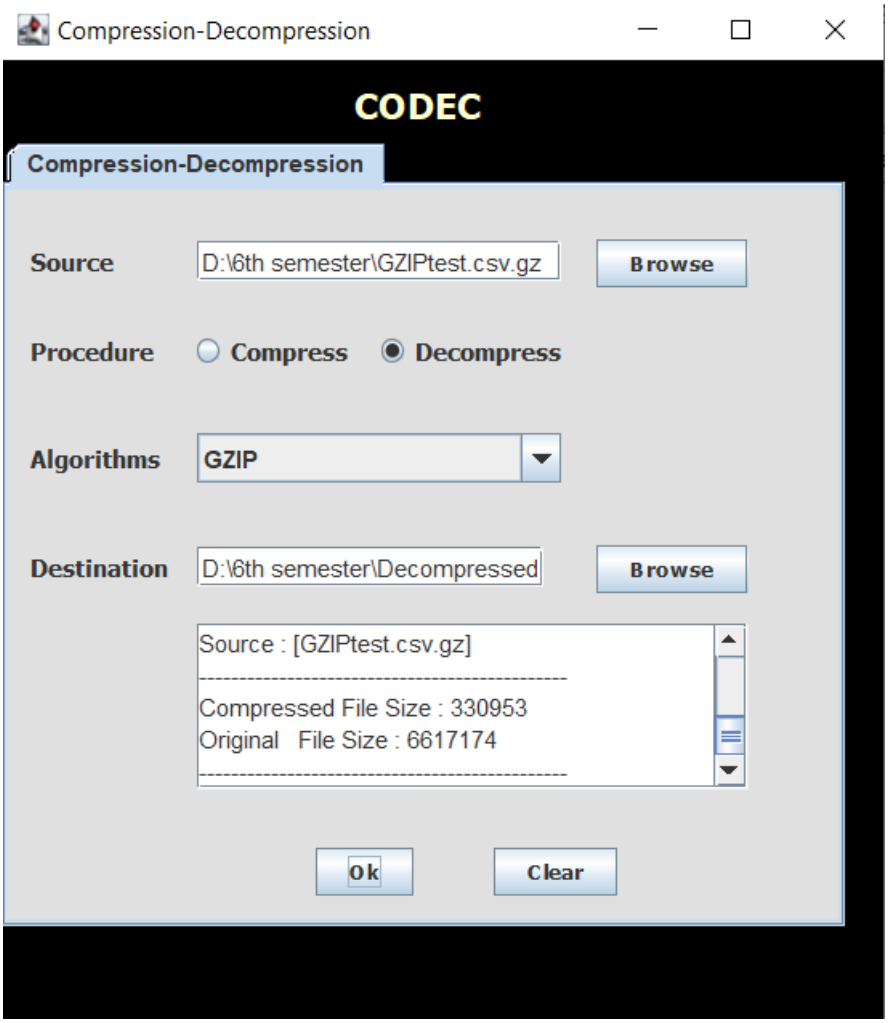


Figure f: Decompression using GZIP

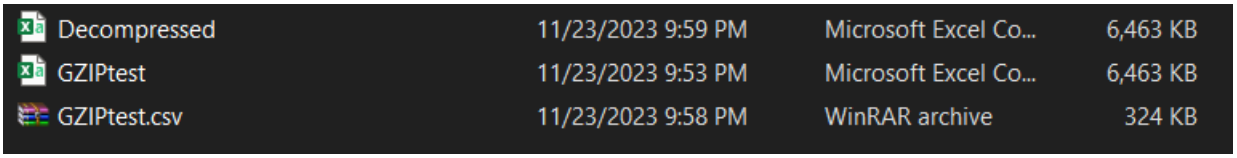


Figure g: Conversion of decompressed file

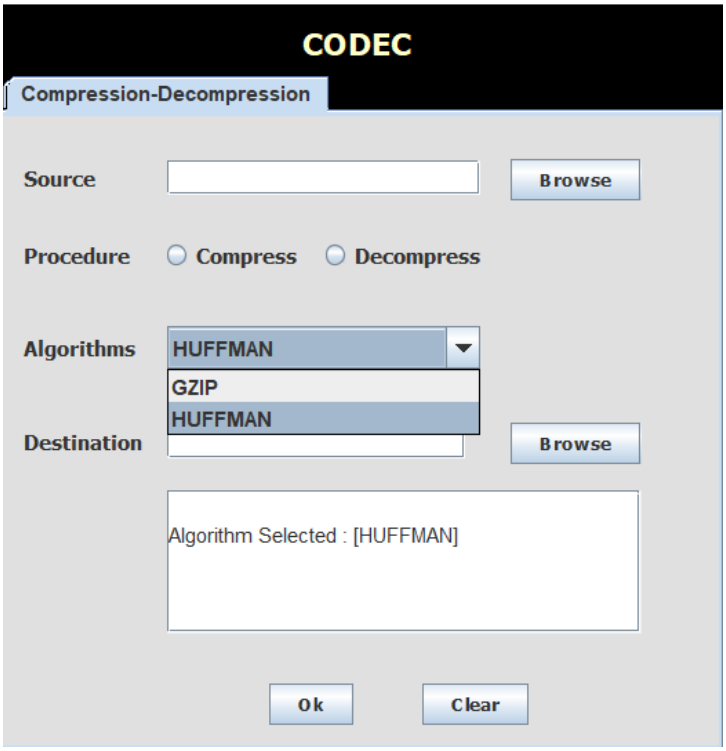


Figure h: Changing the algorithm

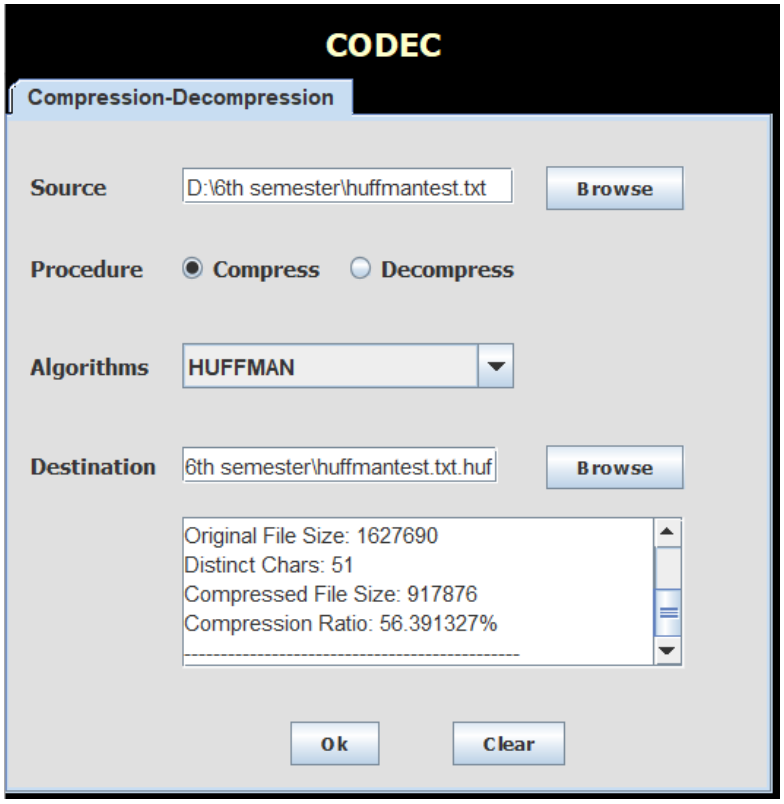


Figure i: Compressing the file using HUFFMAN

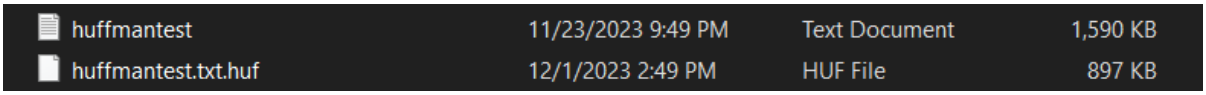


Figure j: Comparison of Size

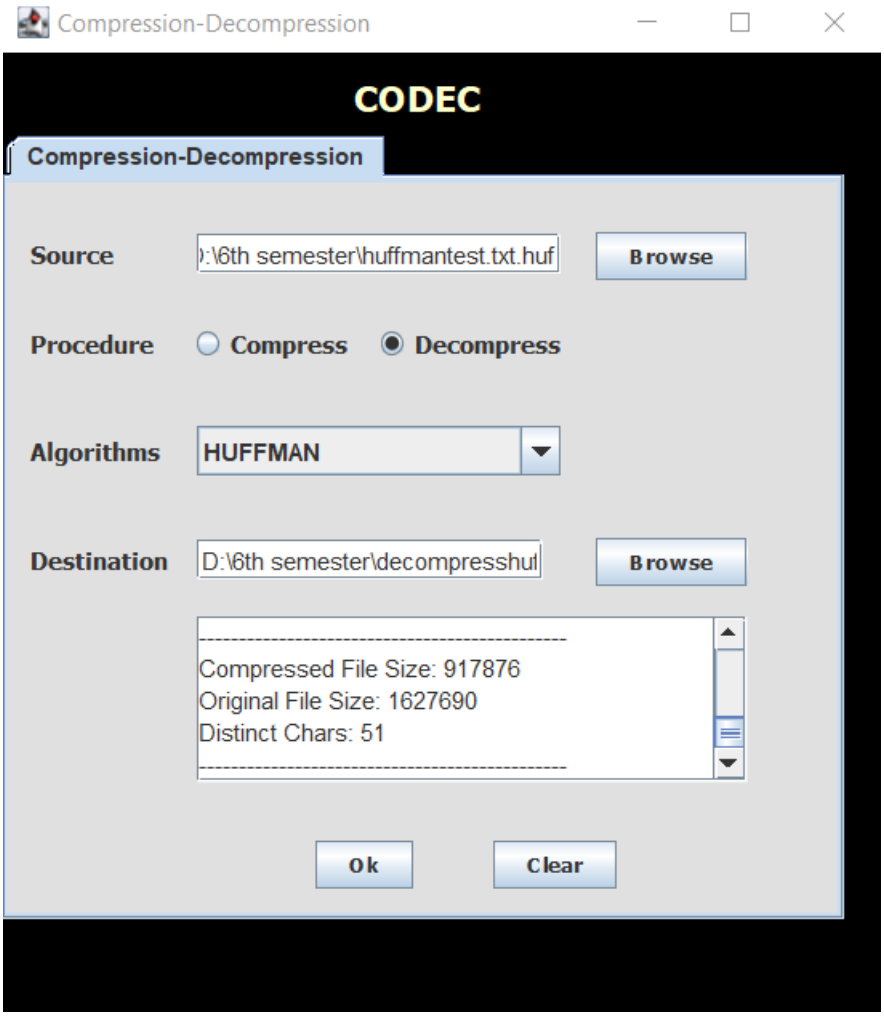


Figure k: Decompression using HUFFMAN

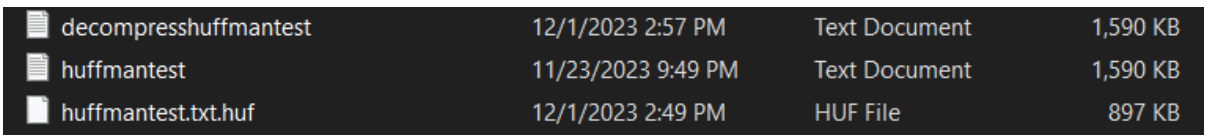


Figure l: Comparison of decompressed file